Chapter 05
Network Layer

Control Plane

Network Layers

Two important planes & functions

- **Data Plane**: actions inside each *individual routers* (forwarding)
- **Control Plane**: “coordinated” actions among all the routers (routing)
  - Traditional routing algorithms
    - Centralized: Dijkstra like algorithm (link state)
    - Distributed: Bellman-Ford algorithm (distance vector)
  - Network management (optional)
  - Network configuration (optional)

Routing Algorithms:
How To Find Best Paths
From One Source to Many Destinations

(and build the forwarding table at that source)

Road Navigation

These apps have a ***global map*** of the entire world!
Current traffic conditions

Routers and Routing Table

Three way to populate routing tables:

- Direct connection to another router
- Static routes: manual configuration by network admin
- Dynamic routes: routers learned automatically from other routers

Routing Algorithms (Executed by Routers)

- Goal: compute "good" path<u>s</u>/route<u>s</u> from senders to receivers
  - *Not just a single path!*
  - Good: "cheapest", "fastest", "shortest", "least congested", ...
- Two well-known algorithms
  - Dijkstra like based on **link state**
  - Bellman-Ford based on **distance vector**
- Modeled using a weighted graph
  - Nodes/Vertices are routers
  - Edges are network links
  - Weights are the cost of using (direct) links. Interpretation of cost: time, level of congestion, number of hops, etc.

Dijkstra vs. Bellman Ford

	Dijkstra	**Bellman-Ford**
Execution Mode	Centralized	Decentralized
Information Needed	Each router requires the *complete graph of the network*	Each router needs to know *only its immediate neighbors*
Key Step in **each iteration**	Improve the cost only to neighbors of the best vertex	Improve the cost over all edges in the graph (*can be done in parallel*)
Advantage	Globally Faster	Globally Slower
Local advantage	*None: every node runs the same amount of work*	*Faster: workload of each node depends on number of neighbors*
Limitation	Can't handle negative edges	Can handle negative edges
Used by	OSPF (RFC2328), IS-IS (RFC1195)	RIP (RFC1058), EIGRP (RFC7868)

Link State

(Current) State of a link:

- Is it up or down?
- Its IP address and network mask
- What type of network it is connected to

Dijkstra Link-State Algorithm

- Centralized, requires knowledge of the entire network topology
- Iterative, computes the least cost from a SINGLE source node (u) to ALL other destinations nodes
  - The output can be used as the forwarding table at u
  - Each router runs the Dijkstra algorithm using its own node as the source to compute its forwarding table
- After k iterations, the algorithm knows the least cost path to k destinations
- Route Oscillation is possible when cost is computed based on dynamic properties (such as the current congestion level, the current amount of traffic)

Shortest Path Algorithm (1958)

```
// Shortest paths from source S to all other nodes
function Dijkstra(S) {
    for each vertex v {
        dist[v] = INFINITY
        pred[v] = UNDEFINED
        add v to Unvisited
    }
    dist[S] = 0
    while Unvisited is not empty {
        u = vertex in Unvisited with minimum dist[u]
        remove u from Unvisited
        for each neighbor v of u in Q {
            alt = dist[u] + cost(u,v)
            if alt < dist[v] {
                dist[v] = alt
                pred[v] = u
            }
        }
    }
}
```

Dijkstra Visualization

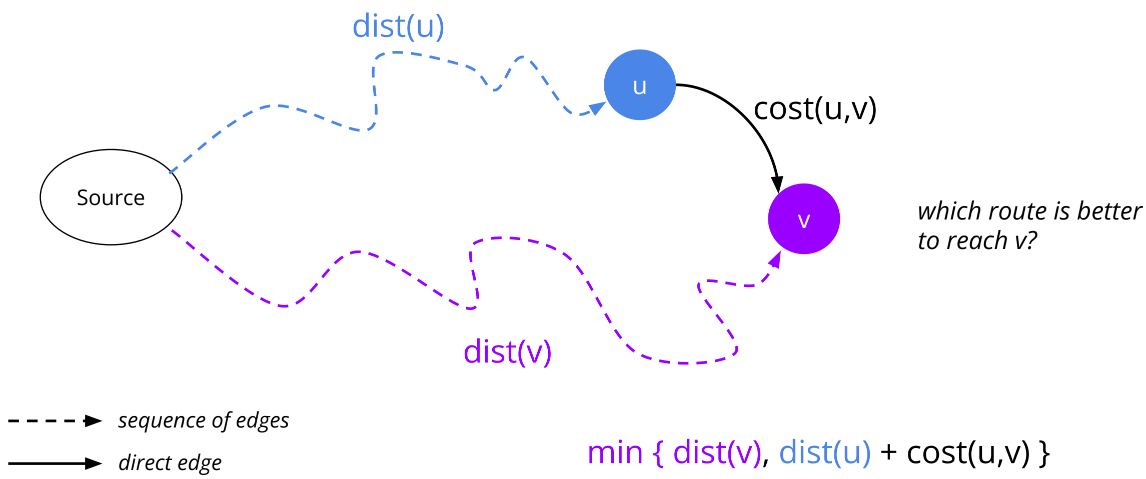
O(E + V log V) or O((E + V) log V)

```
// Compute shortest paths from source S to all other nodes
function BellmanFord(S) {
    for each vertex v {
        dist[v] = INFINITY
        pred[v] = UNDEFINED
    }

    dist[S] = 0
    repeat M-1 times { // M is the number of vertices


        for each edge (u,v) {
            alt = dist[u] + cost(u,v)
            if alt < dist[v] {
                dist[v] = alt
                pred[v] = u
            }
        }
    }
}
```

only this portion runs in each router

O(V × E)

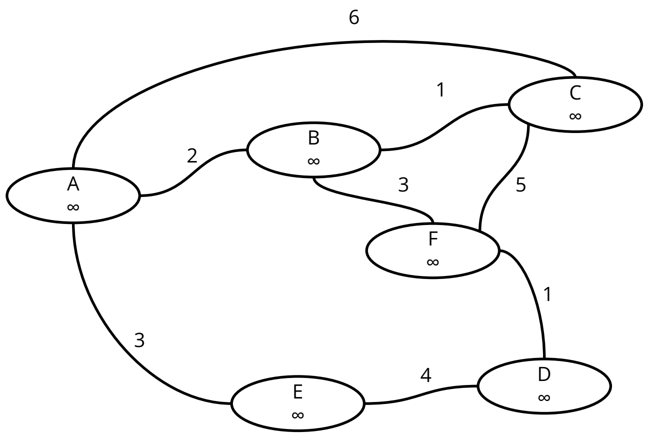
Shortcuts in Dijskstra/Bellman-Ford



Bellman-Ford Algorithm

- Decentralized / distributed algorithm
  - Each node/router computes its best path to the destination ("distance vector") and broadcast this information to all its neighbors
  - Upon receiving a set of distance vectors (from its neighbors), a node updates its best path calculation (and send the updated cost to its neighbors)
- Based on dynamic programming approach
- Iterative
  - At time 0 every node has the distance vector available on at itself
  - At (the end of) time 1 the distance vector of a node has propagated to nodes 1 hop away from that node
  - At (the end of) time k the distance vector of a node has propagated to nodes k hops away from that node

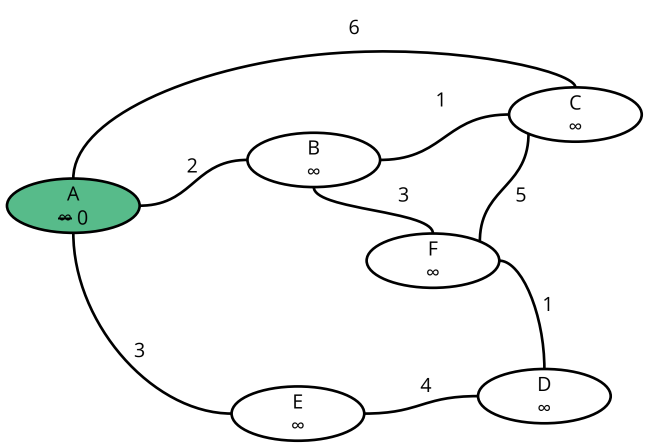
Dijkstra Shortest Path Example



Node	dist	pred
A	∞	?
B	∞	?
C	∞	?
D	∞	?
E	∞	?
F	∞	?

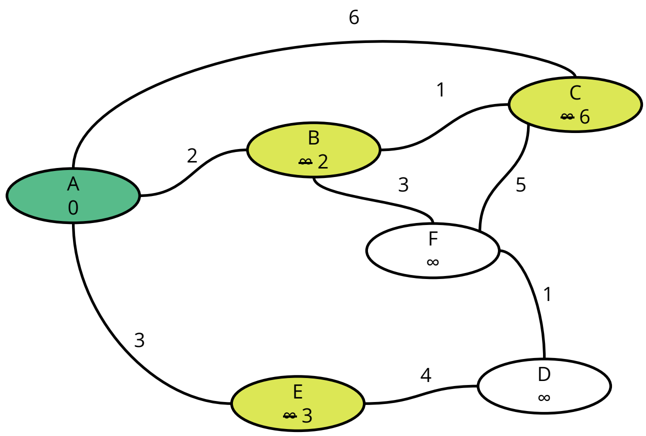
Dijkstra Shortest Path Example

Source A



Node	dist	pred
A	~~∞~~ 0	None
B	∞	?
C	∞	?
D	∞	?
E	∞	?
F	∞	?

Dijkstra Shortest Path Example

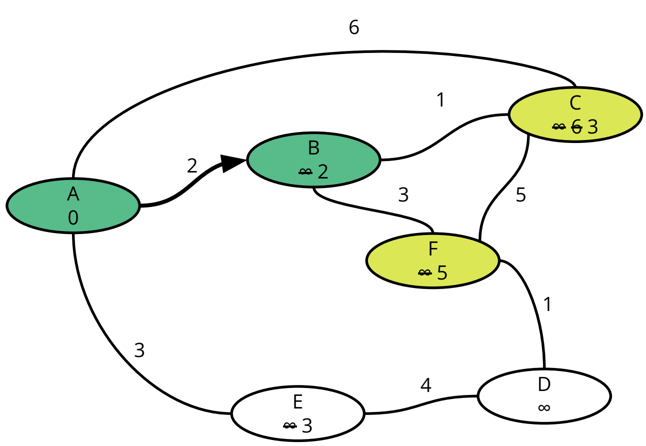


Source A
A *unvisited* neighbors: B, C, E

Node	dist	pred
A	0	None
B	~~∞~~ 2	A
C	~~∞~~ 6	A
D	∞	?
E	~~∞~~ 3	A
F	∞	?

Best *unvisited* neighbor: B

Dijkstra Shortest Path Example

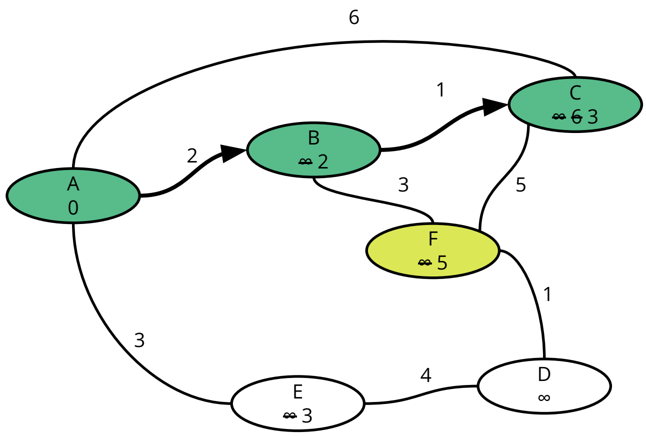


Source A
B *unvisited* neighbor(s): C, F

Node	dist	pred
A	0	None
B	2	A
C	6 ⇒ 3	~~A~~ B
D	∞	?
E	3	A
F	∞ ⇒ 5	B

Best *unvisited* node: C (or E)

Dijkstra Shortest Path Example

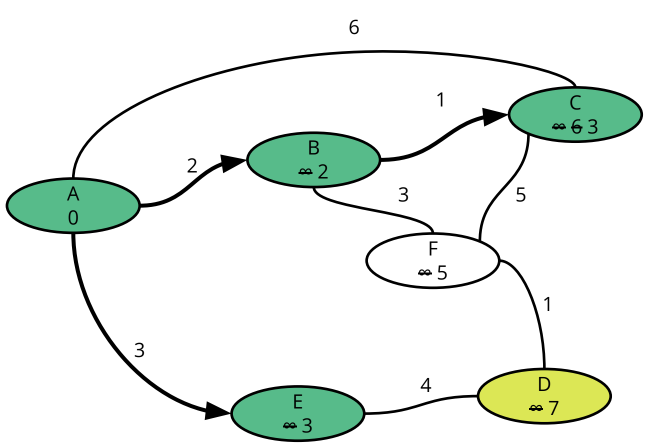


Source A
C *unvisited* neighbor(s): F

Node	dist	pred
A	0	None
B	2	A
C	3	B
D	∞	?
E	3	A
F	5	B

F unchanged
Best *unvisited* node: E

Dijkstra Shortest Path Example

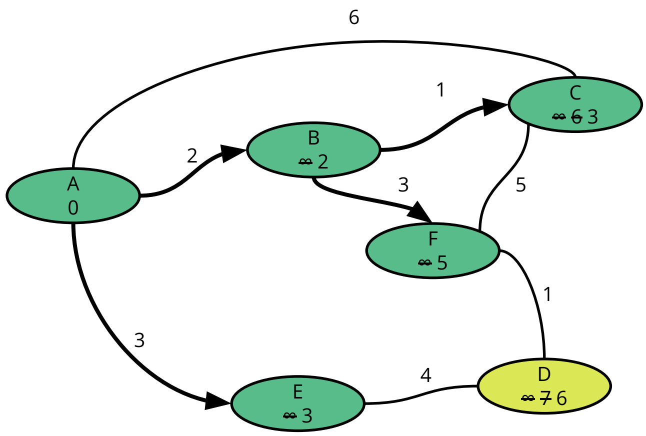


Source A
E *unvisited* neighbor(s): D

Node	dist	pred
A	0	None
B	2	A
C	3	~~A~~ B
D	~~∞~~ ⇒ 7	E
E	3	A
F	5	B

Best *unvisited* node: F

Dijkstra Shortest Path Example

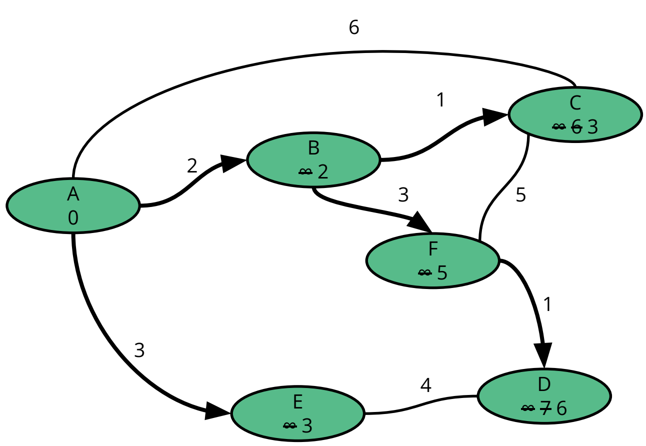


Source A
F *unvisited* neighbor(s): D

Node	dist	pred
A	0	None
B	2	A
C	3	~~A~~ B
D	~~7~~ 6	~~E~~ F
E	3	A
F	5	B

Best *unvisited* node: D

Dijkstra Shortest Path Example

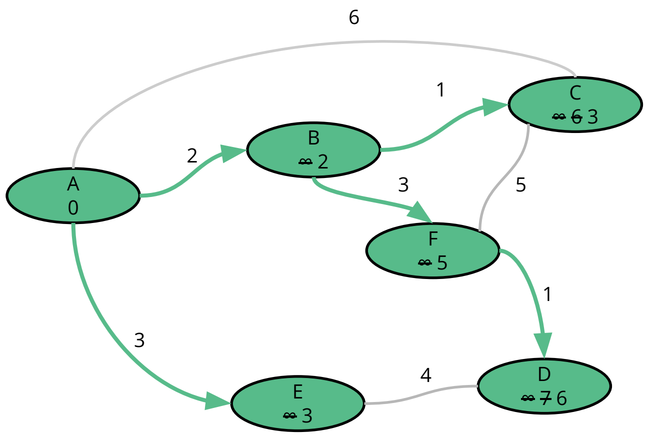


Source A
D *unvisited* neighbor(s): None

Node	dist	pred
A	0	None
B	2	A
C	3	~~A~~ B
D	~~7~~ 6	~~E~~ F
E	3	A
F	5	B

Best *unvisited* node: D

Dijkstra Shortest Path Example

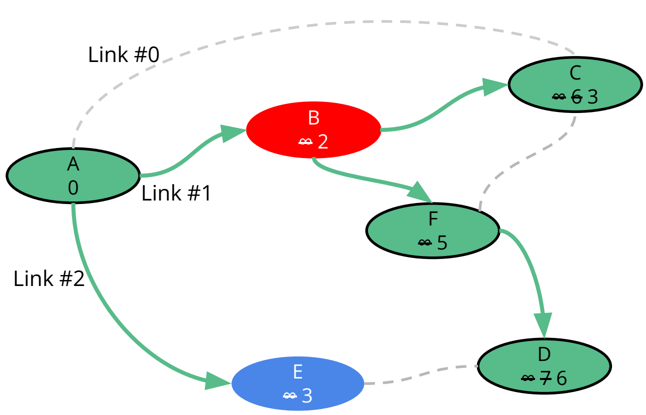


Source A
D *unvisited* neighbor(s): None

Node	dist	pred
A	0	None
B	2	A
C	3	~~A~~ B
D	~~7~~ 6	~~E~~ F
E	3	A
F	5	B

Best *unvisited* node: None

<u>Forwarding</u> Table for Node A



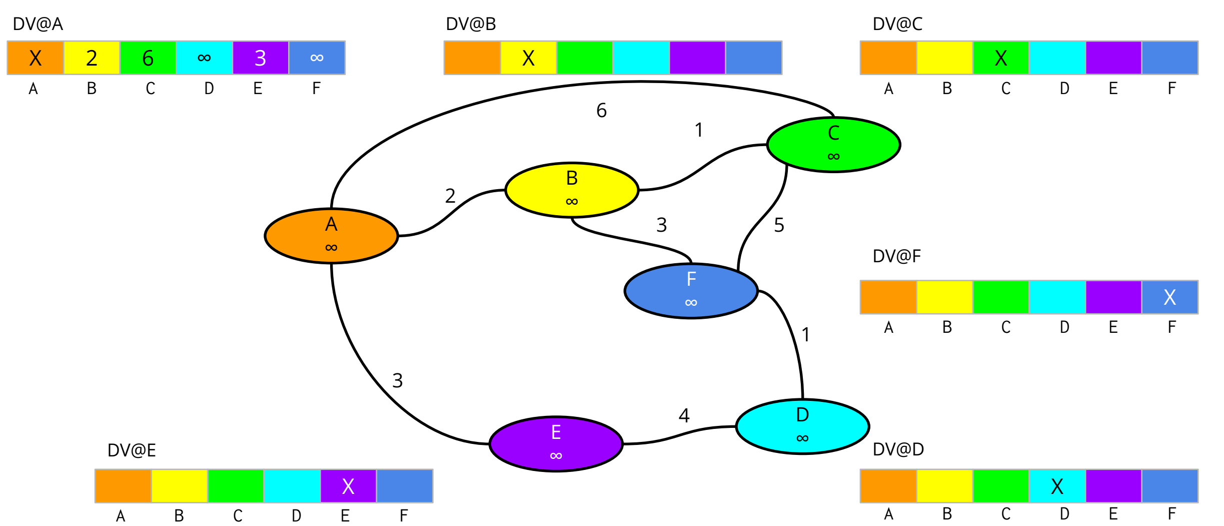
Dest	Path	Output Link
B	A ⇒ **B**	**#1**
C	A ⇒ **B** ⇒ C	**#1**
D	A ⇒ **B** ⇒ F ⇒ D	**#1**
E	A ⇒ **E**	**#2**
F	A ⇒ **B** ⇒ F	**#1**

Output link is determined from next-hop from A

Bellman-Ford Algorithm

Bellman-Ford: Initial Distance Vectors

Partially Complete



Bellman-Ford: Initial Distance Vectors

DV@A

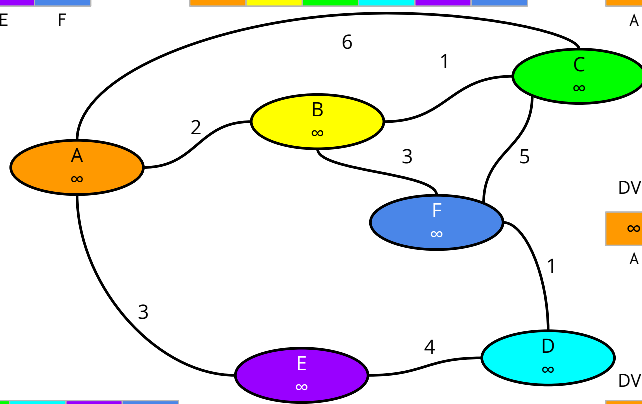
A	B	C	D	E	F
X	2	6	∞	3	∞

DV@B

A	B	C	D	E	F
2	X	1	∞	∞	3

DV@C

A	B	C	D	E	F
6	1	X	∞	∞	5



DV@F

A	B	C	D	E	F
∞	3	5	1	∞	X

DV@E

A	B	C	D	E	F
3	∞	∞	4	X	∞

DV@D

A	B	C	D	E	F
∞	∞	∞	X	4	1

Bellman-Ford: B receives update from A

DV from A (incoming)

A	B	C	D	E	F
X	2	6	∞	3	∞

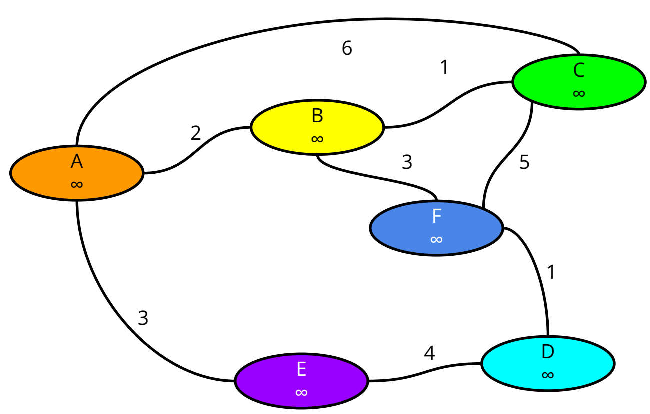
DV@B (current)

A	B	C	D	E	F
2	X	1	∞	∞	3

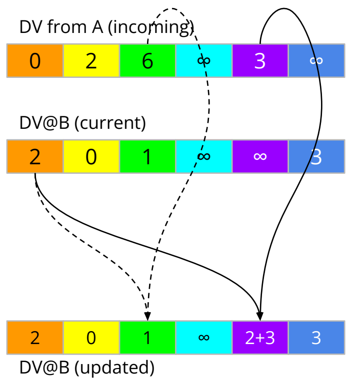
DV@B (updated)

A	B	C	D	E	F
2	X				

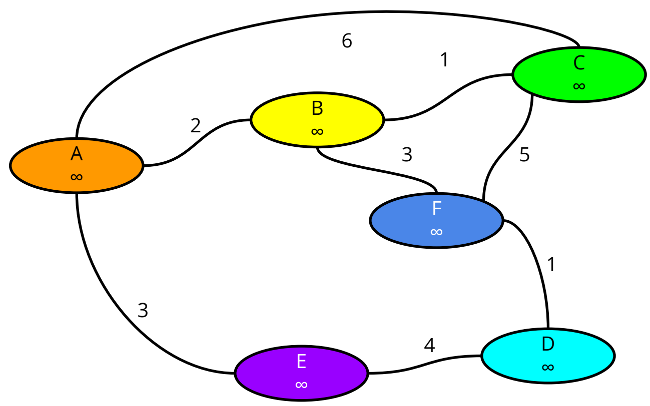
B shall update only info $\{A,B\}^{comp}$



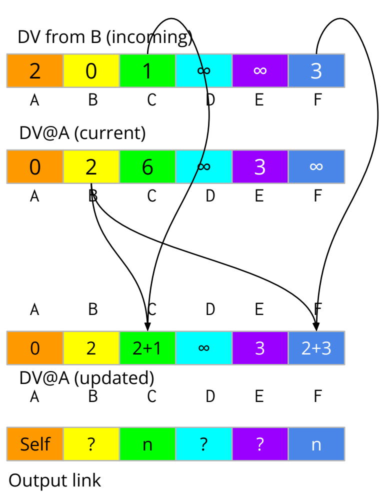
Bellman-Ford: B receives update from A



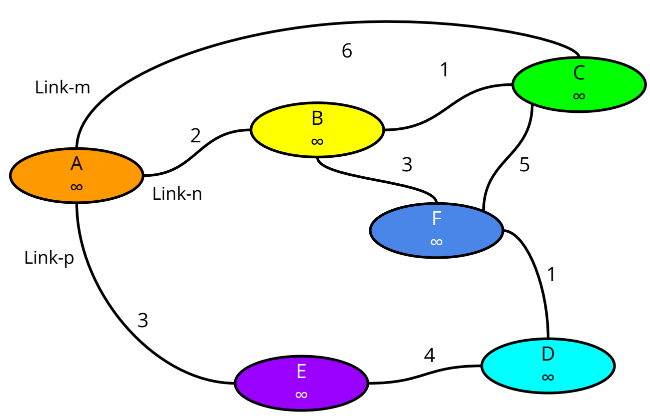
A shall update only info {A,B}^{comp}



Bellman-Ford: A receives update from B (thru Link-n)

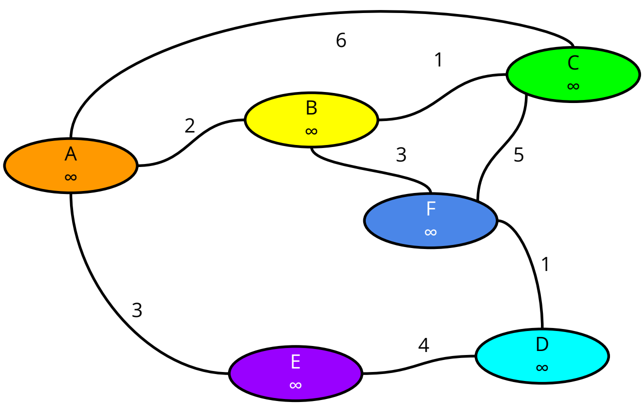


A shall update only info {A,B}^{comp}



Bellman-Ford: C receives update from B (group exercise)

B shall update only info {B,C}^{comp}



DV@B

A	B	C	D	E	F
2	0	1	∞	∞	3

DV@C (current)

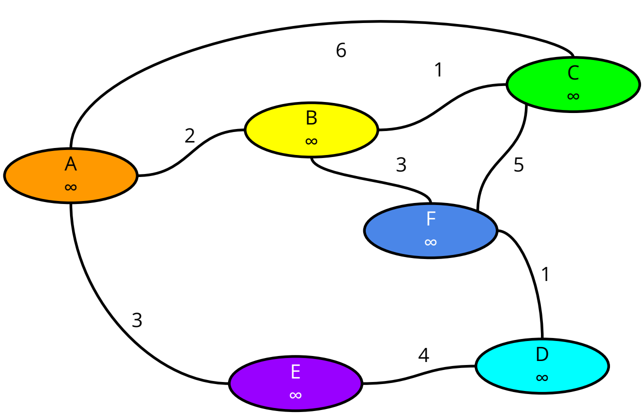
A	B	C	D	E	F
6	1	0	∞	∞	5

A	B	C	D	E	F
	1	0			

DV@C (updated)

Bellman-Ford: C receives update from B

B shall update only info {B,C}^{comp}



DV@B

A	B	C	D	E	F
2	0	1	∞	∞	3

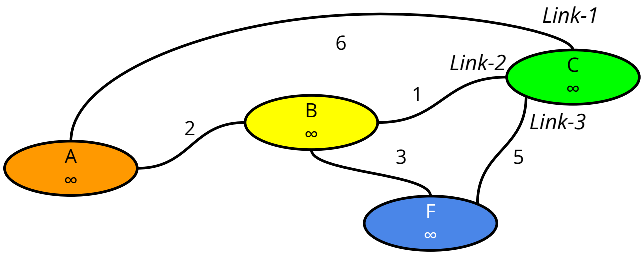
DV@C (current)

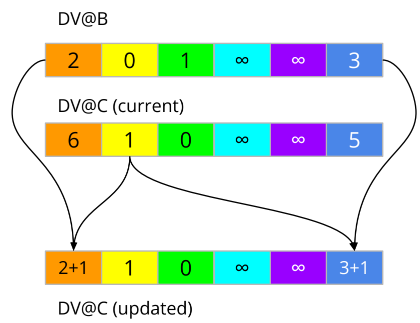
A	B	C	D	E	F
6	1	0	∞	∞	5

A	B	C	D	E	F
2+1	1	0	∞	∞	3+1

DV@C (updated)

Bellman-Ford: Building Forwarding Table



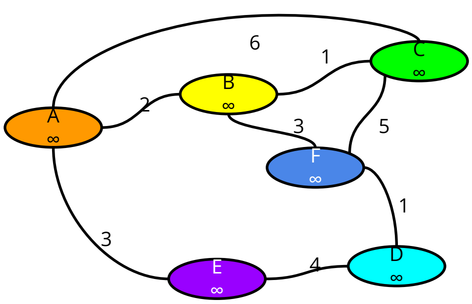


Current Forwarding Table at C

Dest	Output Link (current)	Output Link (updated)
A	Link-1	Link-2
F	Link-3	Link-2

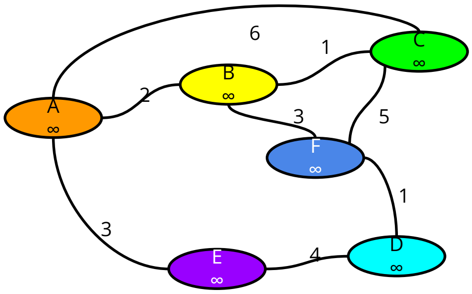
B sent update to C *via* *Link-2*

Bellman-Ford: Step 1 propagate A ⇒ (B,C,E)



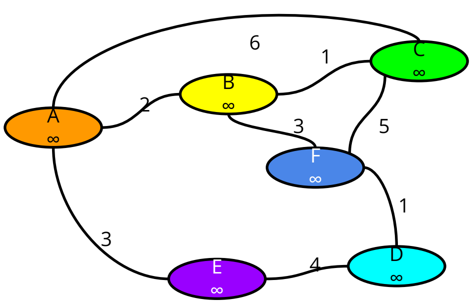
	A	B	C	D	E	F						
DV@A	0	2	6	∞	3	∞						
DV@B	2	0	1	∞	∞	3	2	0	1	∞	5	3
DV@C	6	1	0	∞	∞	5	6	1	0	∞	9	5
DV@D	∞	∞	∞	0	4	1						
DV@E	3	∞	∞	4	0	∞	3	5	9	4	0	∞
DV@F	∞	3	5	1	∞	0						
	A	B	C	D	E	F						

Bellman-Ford: Step 2a propagate from B ⇒ (A,C,F)



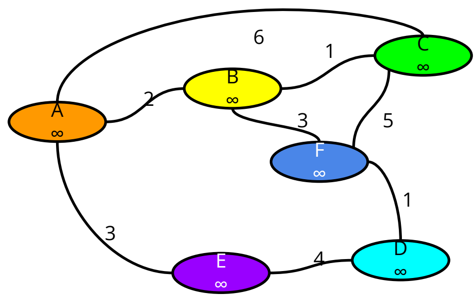
	A	B	C	D	E	F						
DV@A	0	2	6	∞	3	∞	0	2	3	∞	3	5
DV@B	2	0	1	∞	5	3						
DV@C	6	1	0	∞	9	5	3	1	0	∞	6	4
DV@D	∞	∞	∞	0	4	1						
DV@E	3	5	9	4	0	∞						
DV@F	∞	3	5	1	∞	0	5	3	4	1	8	0
	A	B	C	D	E	F						

Bellman-Ford: Step 2b propagate from C ⇒ (A,B,F)



	A	B	C	D	E	F						
DV@A	0	2	3	∞	3	5	0	2	3	∞	3	5
DV@B	2	0	1	∞	5	3	2	0	1	∞	5	3
DV@C	3	1	0	∞	6	4						
DV@D	∞	∞	∞	0	4	1						
DV@E	3	5	9	4	0	∞						
DV@F	5	3	4	1	8	0	5	3	4	1	8	0
	A	B	C	D	E	F						

Bellman-Ford: Step 2c propagate from E ⇒ (A,D)

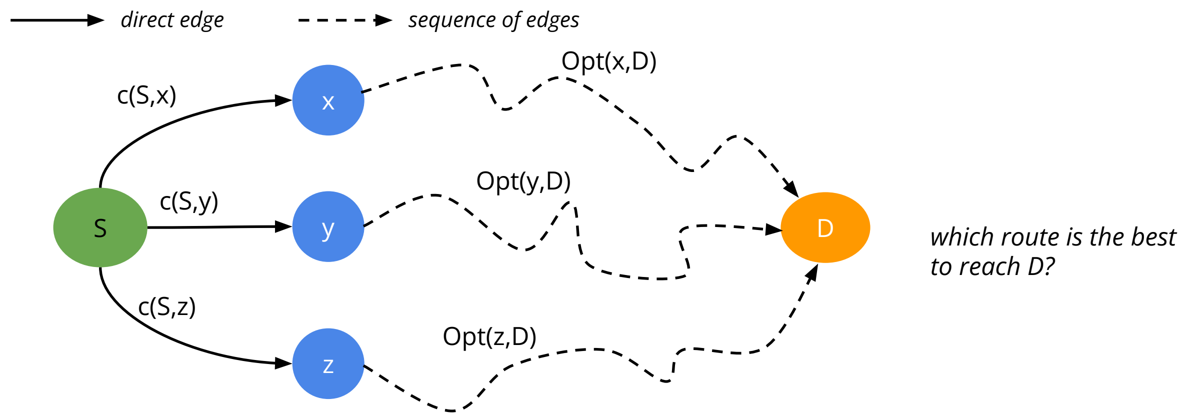


	A	B	C	D	E	F						
DV@A	0	2	3	∞	3	5	0	2	3	7	3	5
DV@B	2	0	1	∞	5	3						
DV@C	3	1	0	∞	6	4						
DV@D	∞	∞	∞	0	4	1	7	9	13	0	4	1
DV@E	3	5	9	4	0	∞						
DV@F	5	3	4	1	8	0						
	A	B	C	D	E	F						

Bellman-Ford: Actions per Node

```
repeat {
    listen for update from immediate neighbors
    recalculate my distanceVector
    readjust my routing table
    if my distanceVector has changed {
        send my distanceVector to my immediate neighbors
    }
}
```

Bellman-Ford: Dynamic Programming Calculation



Opt(S,D) = min { c(S,x) + Opt(x,D), c(S,y) + Opt(y,D), c(S,z) + Opt(z,D) }

Bellman-Ford: Effect of Higher Cost Change

DV@A

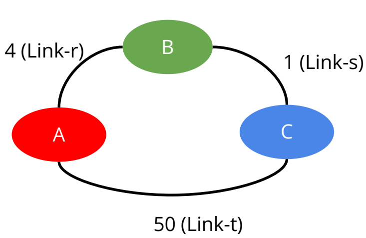
0	4	5

DV@B

4	0	1

DV@C

5	1	0



what if c(B,A) increases?

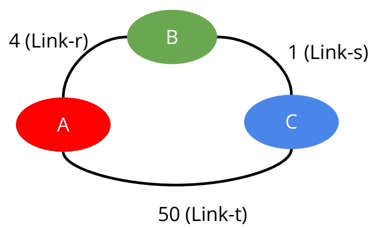
$$\text{Opt}(B,A) = \min\{c(B,A) + \text{Opt}(A,A), c(B,C) + \text{Opt}(C,A)\} = \min\{c(B,A), c(B,C) + \text{Opt}(C,A)\}$$
$$\text{Opt}(C,A) = \min\{c(C,A) + \text{Opt}(A,A), c(C,B) + \text{Opt}(B,A)\} = \min\{c(C,A), c(C,B) + \text{Opt}(B,A)\}$$

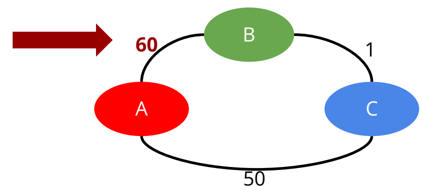
zero

Bellman-Ford: Effect of Cost Increase

DV@A			DV@B			DV@C		
0	4	5	4	0	1	5	1	0

DV@A (after)			DV@B (after)			DV@C		
0	60	5	60	0	1	5	1	0





$$\text{Opt}(B, A) = \min\{4 \longrightarrow 60, 1 + \text{Opt}(C, A)\}$$
$$\text{Opt}(C, A) = \min\{50, 1 + \text{Opt}(B, A)\}$$

Bellman-Ford Issue: "Counting to Infinity"

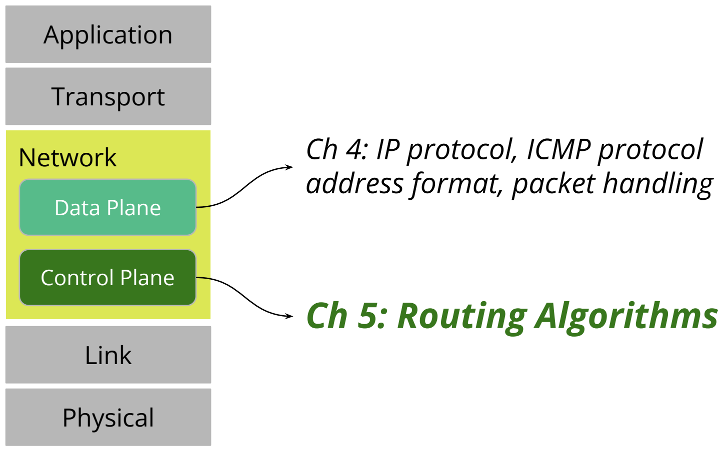
Initial values: **c**(B,A) = 4 *(typo in printout)*
Opt(C,A) = 5
What if c(B,A) changes 4 ⇒ 60 ?

4 ⇒ 6 ⇒ ... ⇒ 48 ⇒ 50 ⇒ 51   4 ⇒ 60   5⇒ 7 ⇒ ... ⇒ 49 ⇒ 50

$$\text{Opt}(B, A) = \min\{c(B, A), 1 + \text{Opt}(C, A)\}$$
$$\text{Opt}(C, A) = \min\{50, 1 + \text{Opt}(B, A)\}$$

5⇒ 7 ⇒ ... ⇒ 49 ⇒ 50   4 ⇒ 6 ⇒ ... ⇒ 48 ⇒ 50 ⇒ 51

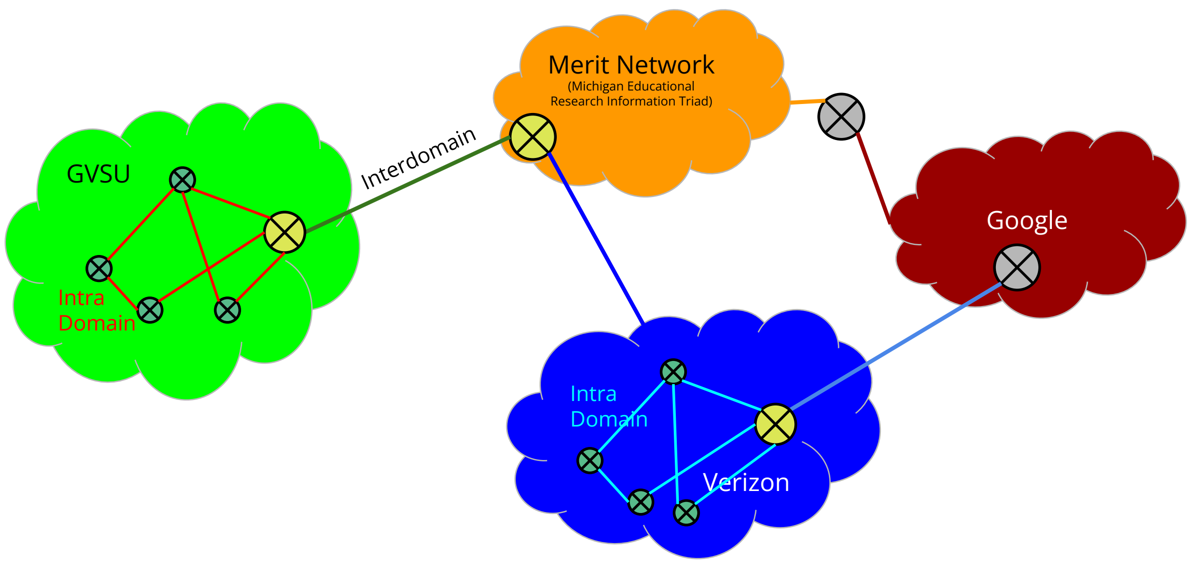
Network Layer: Data and Control Planes



Implementation Issues

- Both Dijkstra & Bellman-Ford algorithms require routers to exchange information to neighbors
- **Issue**: Running either algorithm on a **huge network** is NOT scalable
- **Solution**
  - split the network into groups/domain/regions/autonomous systems
    - *typically* one AS per organization
      - GVSU Allendale, GVSU Pew Campus, AT&T, Comcast, Verizon, etc.
  - distinguish between intra-domain routing and inter-domain routing
    - Intra-domain: GVSU Wifi (in MAK) and GVSU Wifi (in Fieldhouse)
    - Inter domain: Verizon and GVSU network

Intra-Domain vs Inter-Domain



Intra Domain Routing

Protocol	Standard	Algorithm	Note
RIP	RFC1723 (1994)	Bellman-Ford	Limited to 15 hops
EIGRP	RFC7868 (2016)	Bellman-Ford	
OSPF	RFC2328 (1998: IPv4) RFC5340 (2008: IPv6)	Dijkstra	
IS-IS	RFC1195 (1990)	Dijkstra	

IS = intermediate system

EIGRP
Enhanced Interior Gateway Routing Protocol
(Within One Autonomous System)

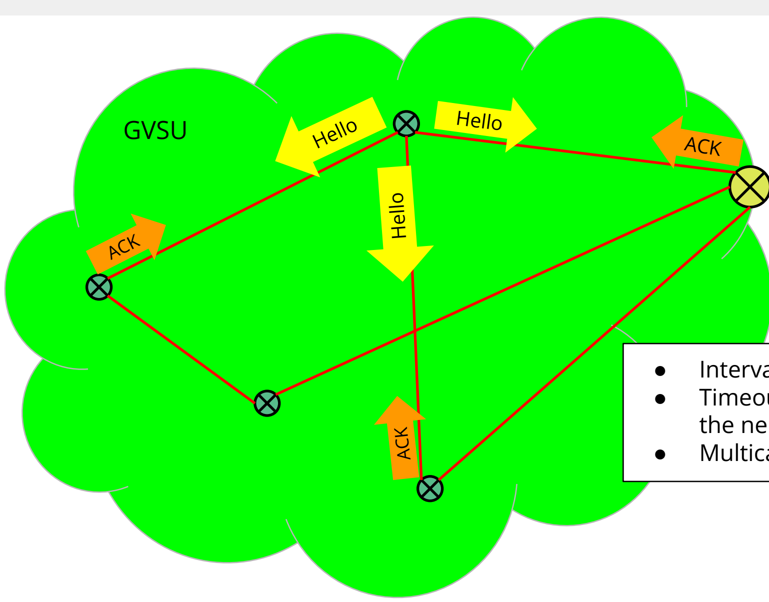
EIGRP

- Based on Distance Vector approach in Bellman-Ford
- Developed by Cisco
- DUAL Algorithm: **D**iffusing **U**pdate **AL**gorithm: loop free diffused computation of a routing table
  - An improvement from basic Bellman-Ford algorithm

EIGRP Messages

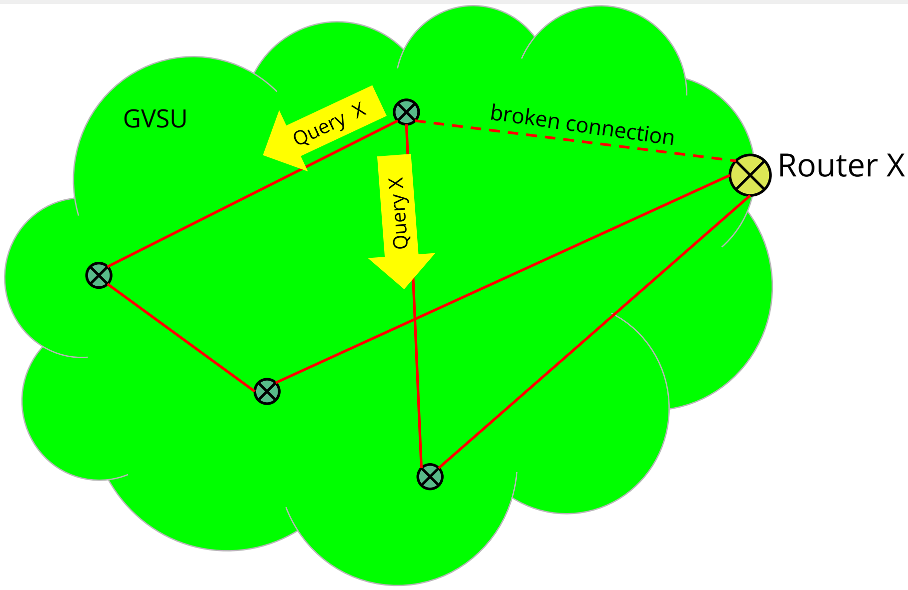
Message	Description	Notes
HELLO	“I’m (still) alive, and will stay alive for N seconds” “Do you want to be my neighbor?” “Yes I’d like to be your neighbor”	Periodically sent by each router
UPDATE	“I’m sharing what I know about destinations X, Y, Z”	
QUERY	“Do you know how to reach destination W?”	Sent out when a router detects that W is no longer alive
REPLY	“Yes this is how to reach W” (In response to QUERY) “No, I don’t”	

EIGRP: HELLO (“I’m alive”)



- Interval: every 5 seconds (or 60 seconds)
- Timeout (3x interval): wait for ACK before the neighbor is consider “dead”
- Multicast to 224.0.0.10

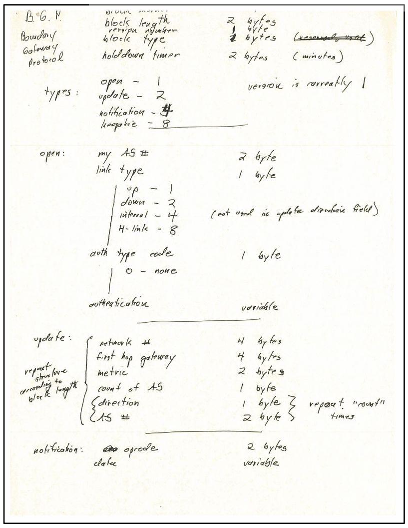
EIGRP: QUERY ("I've not heard from X. Do you know X?")

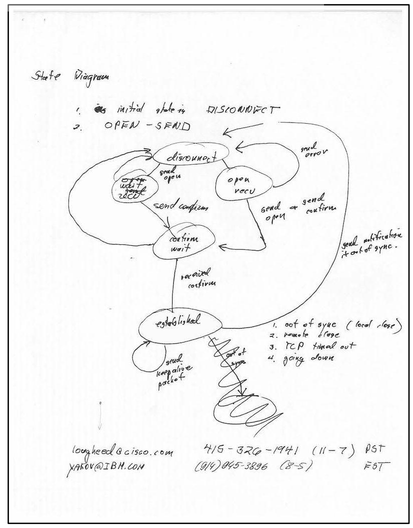


Inter-Domain Routing
Border Gateway Protocol (BGP)

BGP on two napkins over a conference lunch break

RFC1267 Kirk Lougheed & Yakov Rekhter (Oct 1991)





BGP

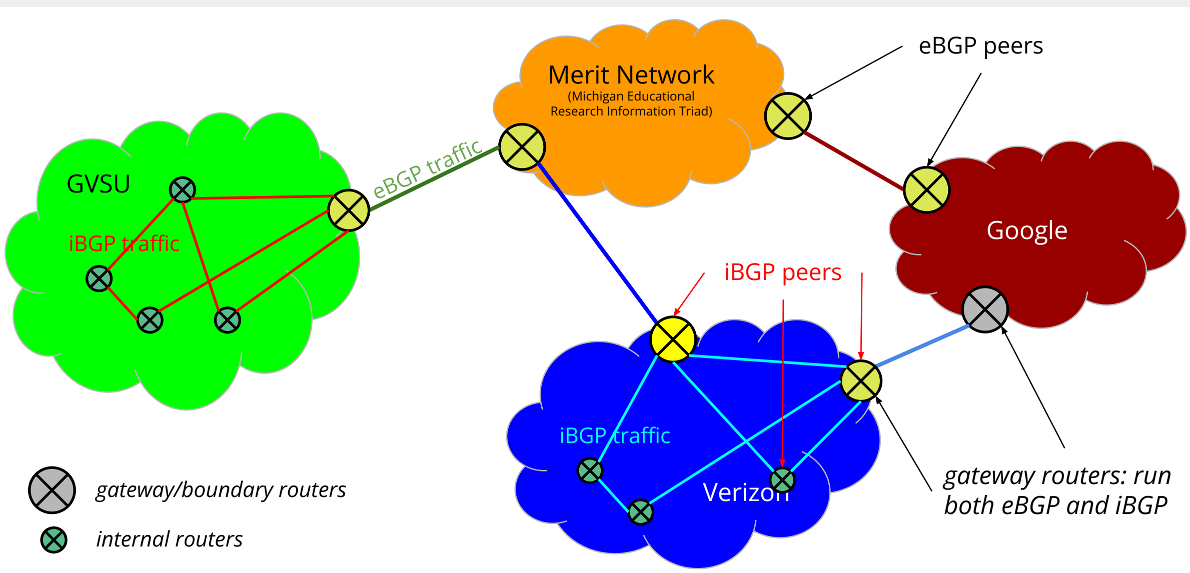
- BGP-3 RFC1267 (Oct 1991) “BGP on Napkins”
- Application of BGP RFC1772 (Mar 1995)
- BGP-4 RFC 4271 (Jan 2006)
- Built on-top of TCP (Port 179)
- Inter Autonomous System Routing Protocol
  - Autonomous System (AS) = “Domains”
  - **Peers**: two autonomous systems that exchange BGP route information

BGP: eBGP + iBGP

Like a coin, BGP has two sides

- Exter(nal|ior) (eBGP) : object reachability information from neighboring autonomous systems ("AS")
- Inter(nal|ior) (iBGP): propagate reachability information to all the routers within **one** autonomous system, which can be managed using
  - IS-to-IS (1990, based on Dijkstra)
  - RIP (1994, based on Bellman-Ford)
  - OSPF (1998, based on Dijkstra)
  - EIGRP (2016, based on Bellman-Ford)
- iBGP is the **prerequisite** to eBGP
  - *eBGP can work properly only after iBGP has been established*

BGP: External BGP & Internal BGP & Peers



BGP: Autonomous System Path Vector & Routing Policy

- Distance vectors in Bellman-Ford algorithm contain **numbers** representing cost to reach **particular nodes**
- Vectors advertised in BGP contain *path to reach a particular AS*
  - The "nodes" in each path is an autonomous system (**group of nodes in CIDR notation**)
  - Hence BGP is also called "**Path Vector**" protocol
- A router can decide whether path details in incoming advertisement will be *re-advertised* to its neighbors or *filtered out* altogether
  - Real world examples
    - Verizon network may **not** want to carry transit traffic from AT&T
    - Verizon customers may have to pay ROAMING charges when their traffic are routed via other provider

Bellman-Ford Distance Vectors vs. BGP Path Vectors

	Distance Vector	**Path Vector**
Used by	Bellman-Ford	(e)BGP
Recorded value	(Best) distance to routers	(Best) Path to autonomous systems
Operation	Distances are added and minimized	During *domain advertisement*, AS names are **appended** (or prepended) to the path

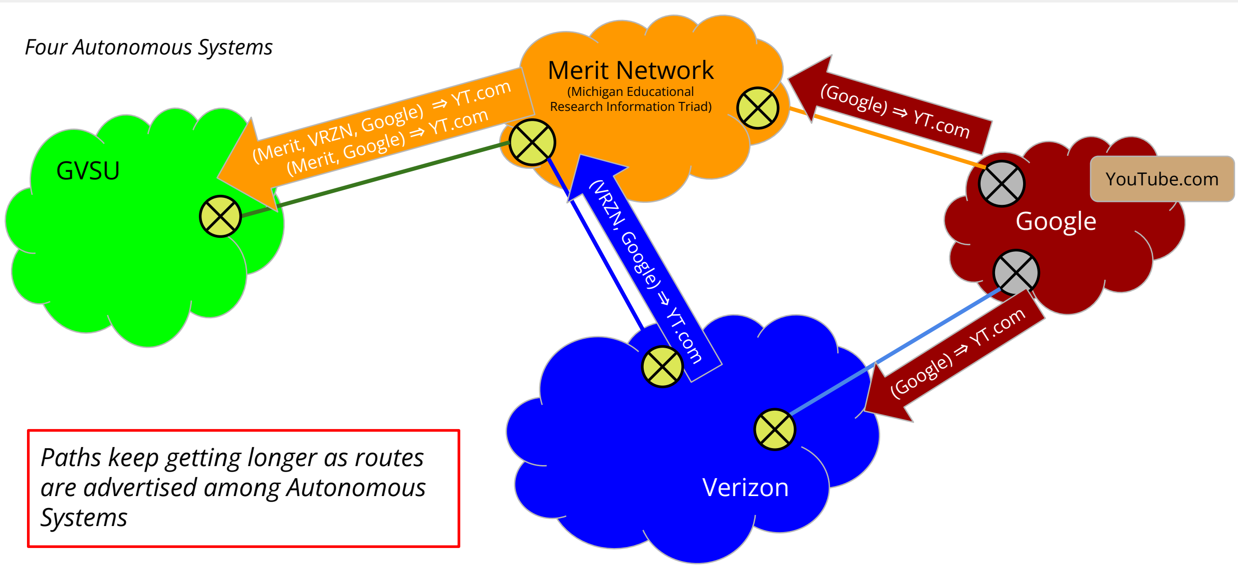
BGP Route Information Details/Attributes

Details included in route announcement:

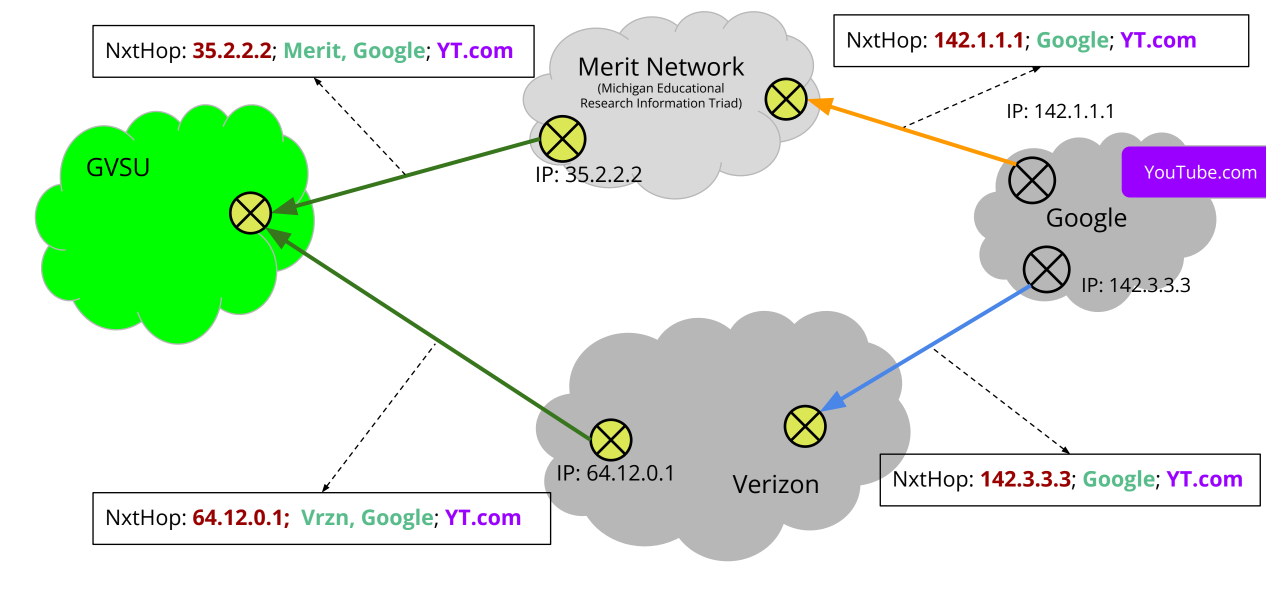
- AS identifier (of the announcer)
- **AS-Path: list of intermediate ASes** to destination
  - "shorter" sequence is preferred
- **Next-Hop**: The IP address of the router of the annoucer eBGP (the "origin AS")
  - "shorter distance" is preferred
- Age of route ("older" is preferred)

eBGP: **AS Path** Advertisement from Google to GVSU

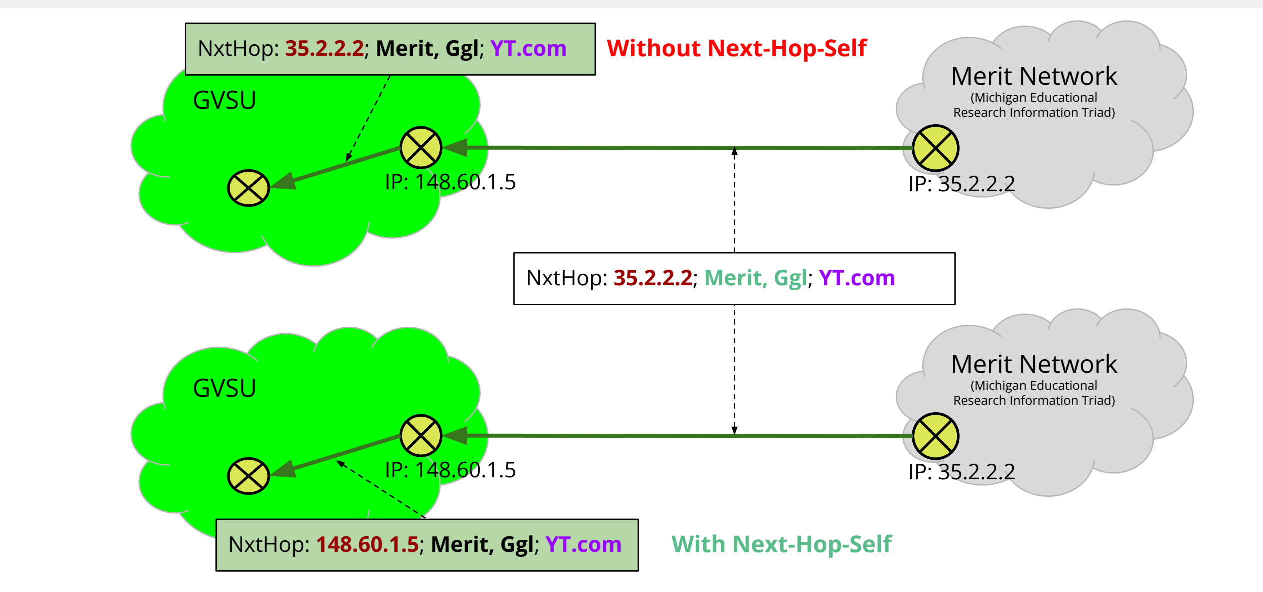
Four Autonomous Systems



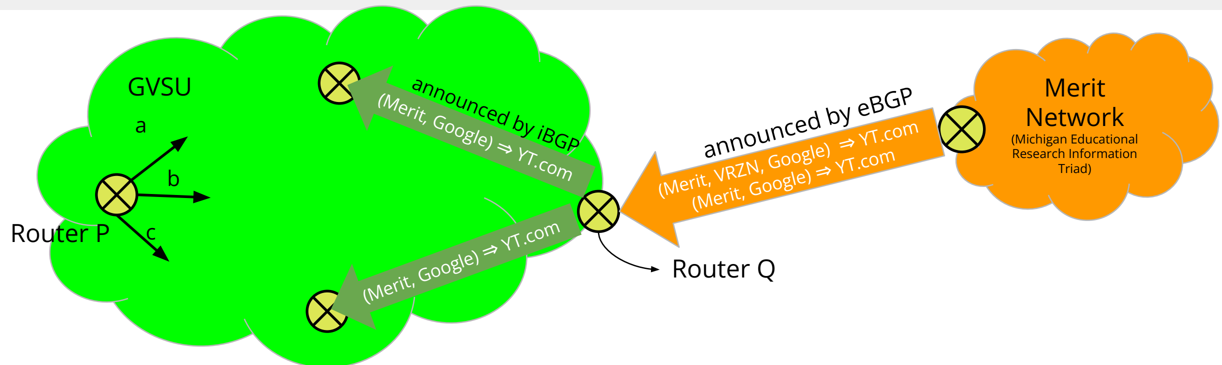
eBGP Routes: Next-Hop



eBGP Next-Hop ⇒ iBGP Next-Hop-Self



eBGP + iBGP Collaboration



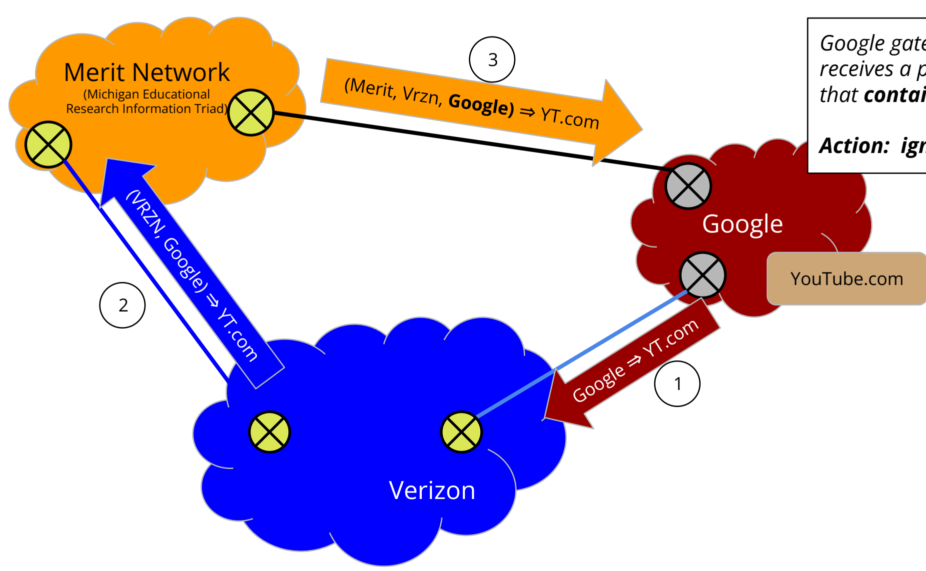
Destination	Output Link	Discovered by
Router Q	c	OSPF, EIGRP, RIP...

Inside Router P

→

Destination	Output Link	Discovered by
Router Q	c	OSPF, EIGRP, RIP...
YT.com	c	iBGP

eBGP: **Loop Detection** in Path Advertisement



Google gateway router receives a path advertisement that ***contains itself.***

Action: ignore

BGP: Best Route Selection Criteria

Order of preference

1. Prefer LOCAL routes (those advertised by other routers in the same AS)
2. Prefer shorter AS path

Route: **35.2.2.2**; **Vrzn, Merit, Google**; **YT.com**	❌: Longer AS Path (three)
Route: **35.2.2.2**; **Vrzn, Google**; **YT.com**	✔: Shorter AS Path (two)

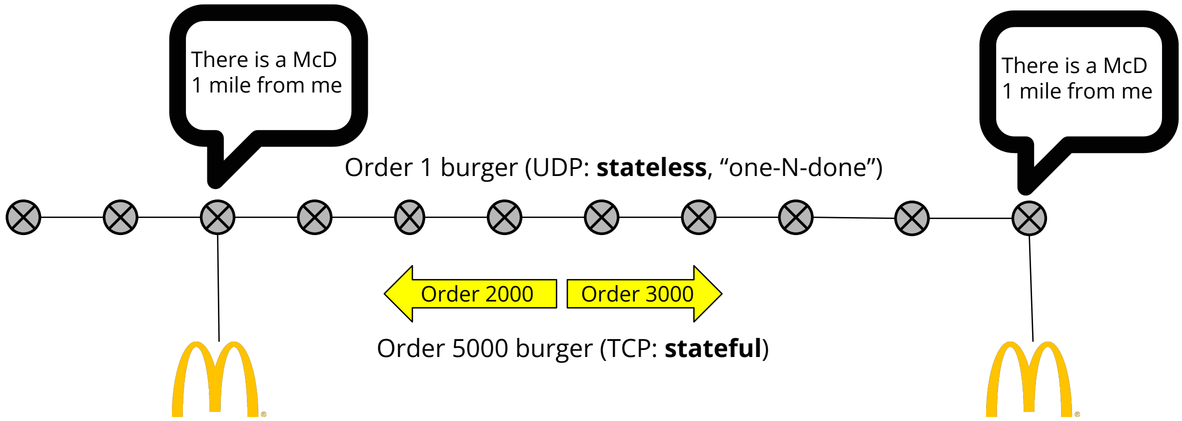
3. Among the paths with the same AS Path length, prefer closer Next Hop

Route: **35.2.2.2**; **Vrzn, Google**; **YT.com**	*Choose route with shorter cost*
Route: **35.5.5.5**; **Vrzn, Google**; **YT.com**	costTo(35.2.2.2) vs. costTo(35.5.5.5)

IP AnyCast (RFC1546)

Delivery Mode	Recipients
Unicast	one (at a specific IP)
Multicast	selected many (those who are “invited”)
Broadcast	many (anyone)
Anycast	the “best” one (IP address does not matter), *usually as the result of BGP algorithm*

AnyCast to McDonald



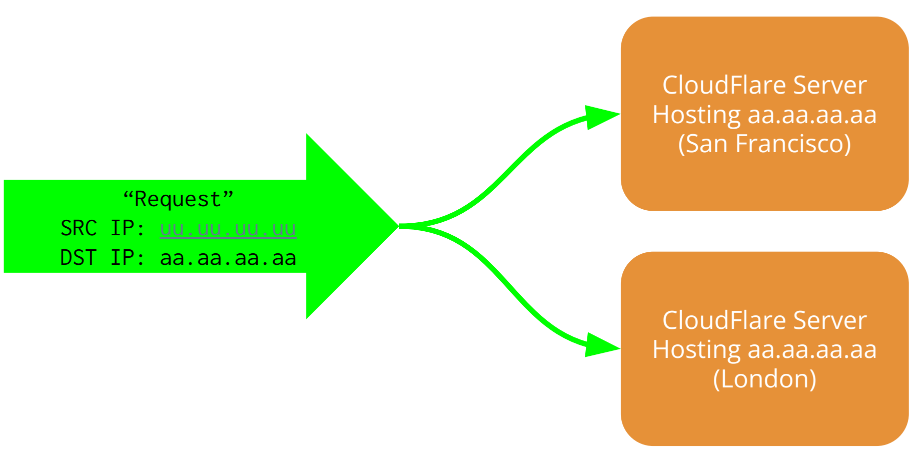
Practical Use of AnyCast

- Content Distribution Network
  - Contents are fetched from a geographic location location closest to the user
- DNS root servers

Case Studies

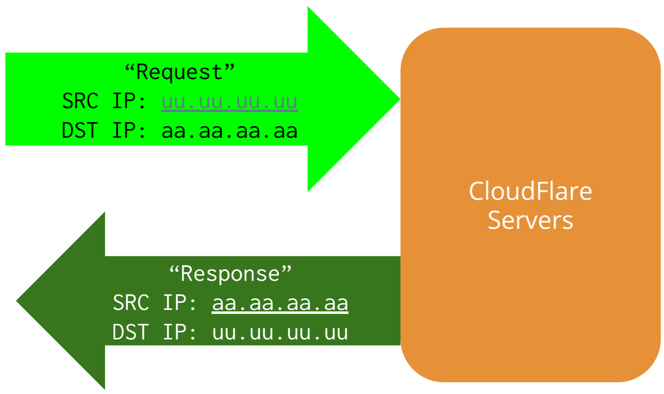
- CloudFlare Routing Solution
  - AnyCast routing
    - UniCast: One Machine, One IP
    - AnyCast: Many Machines, One IP
  - Cloudflare Servers Don't Own IP anymore
    - Ingress Traffic
    - Egress Traffic

CloudFlare Servers: One IP used by many machines



uu.uu.uu.uu = UniCast address
aa.aa.aa.aa = AnyCast address (*managed by CloudFlare*)

CloudFlare: Ingress Traffic

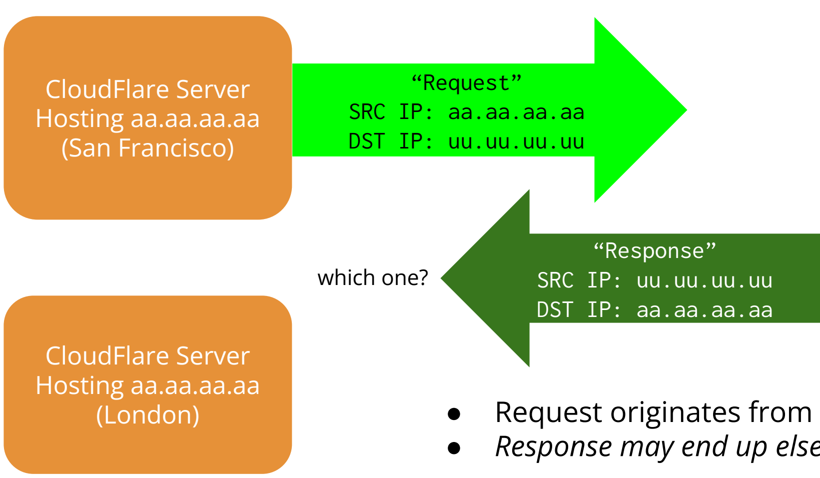


- Packets are routed by their destination
- Incoming packets (**requests**) are routed using anycast address
- Outgoing packets (**responses**) are routed using unicast address
- Responses are guaranteed to reach a unique host

uu.uu.uu.uu = UniCast address
aa.aa.aa.aa = AnyCast address (*managed by CloudFlare*)

CloudFlare: Egress Traffic

uu.uu.uu.uu = UniCast address
aa.aa.aa.aa = AnyCast address



- Request originates from San Francisco (SFO)
- *Response may end up elsewhere*

CloudFlare: Egress Traffic

uu.uu.uu.uu = UniCast address
aa.aa.aa.aa = AnyCast address

CloudFlare Server Hosting aa.aa.aa.aa (San Francisco)

"Request"
SRC IP: aa.aa.aa.aa
DST IP: uu.uu.uu.uu

Solution
- *Source IP address cannot use anycast source IP addr (of the virtual host)*
- *Source IP address is the unicast IP addr of the server at specific location*

CloudFlare Server Hosting aa.aa.aa.aa (London)

New Problem
- To replicate the virtual host aa.aa.aa.aa on N locations, CloudFlare must allocate N unique IP addresses (**EXPENSIVE**)!

New Solution
- Just use ONE IP per virtual host per location
- Associate each location with a range of port numbers

CloudFlare: Egress Traffic Port Range

hh.hh.hh.hh = Virtual Host UniCast address
uu.uu.uu.uu = Destination unicast address